UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Generación de interfaces gráficas por medio de anotaciones en Java

Autor
Oscar Mellado González

Tutor
Juan de Lara Jaramillo

Mayo 2015

Generación de interfaces gráficas por medio de anotaciones en Java

AUTOR: Oscar Mellado González

TUTOR: Juan de Lara Jaramillo

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio 2015

Resumen

La sociedad actual está completamente informatizada de tal manera que nunca lo habríamos imaginado. Cualquier pequeño negocio, cualquier empresa o cualquier multinacional necesita estar informatizada para desempeñar su función. En este proceso de informatización entran en juego los programas desarrollados de cara a usuarios "básicos", entendiendo como básicos a los usuarios sin formación en informática. Todos estos programas necesitan una apariencia visual atractiva e intuitiva para que los usuarios la utilicen, dicho en un lenguaje más técnico, todos los programas necesitan tener una interfaz gráfica de usuario.

La creación de estas ventanas es un trabajo que a los programadores no les gusta nada, ya que les parece un trabajo muy costoso y que necesita mucho tiempo. Se prefiere dejar esta creación a diseñadores gráficos, pero como normalmente no se puede disponer de uno se ha intentado solucionar este problema de la mejor manera posible. Esta solución es crear un generador de código, que genere las interfaces gráficas.

Por estas razones se ha decido crear el ***APT Processor***. ***APT Processor*** es una herramienta que permite la creación automática de interfaces gráficas de usuario por medio de **Anotaciones** en el lenguaje de programación Java. Con unas pequeñas configuraciones el desarrollador de software tendrá creadas dinámicamente las interfaces gráficas que desee. Estas interfaces gráficas que se generan son bastante sencillas, pero incluyen validaciones de datos y alguna pequeña personalización.

Gracias a la utilización de esta herramienta se podrá ahorrar tiempo en el desarrollo de software de usuario, ya que no será necesario programar las interfaces gráficas, tan solo la funcionalidad del programa.

Palabras clave

Anotaciones, Procesadores de Anotaciones, Interfaces gráficas, Swing, Java, Generación de código.

Abstract

Today's society is completely computerized as we would have never imagined. Every small company, bussiness or multinational needs to be computerized to perform its function. In this computerized process developed programs to "basic" users come on the scene, meaning as "basic" users thoose users without computer knowledge. All these programs need an intuitive and atractive visual appereance for users to use it, in a more technical lenguaje, all programs need to have a graphical user interface.

This windows creation is a work that developers do not like, because we think it is a very expensive work and it takes too much time to performe. We prefere to leave it to Graphic Designers, but as you cannot usually have one, it has been attempting to solve this problem in the best possible way. This solution is to create code generator wich generate graphical interfaces.

For these reasons **APT Processor** has been created. **APT Processor** is a tool that enables automatic creation of graphical user interfaces by Java Annotations. With small configurations, software developer will have dynamically generated graphical interfaces you want. This generated graphical interfaces are very simple, but include data validation and some small customization.

By using this tool, you can save development time, and it will not be necessary to program graphical interfaces, only the functionality of the program.

Keywords

Annotation, Process Anotation, Graphical user interface, Swing, Java, code generation.

Agradecimientos

Quisiera agradecer a mi familia y a mi novia todo el apoyo que me han estado dando desde que comencé mi etapa como universitario, ya que sin su apoyo nunca lo habría conseguido. Me gustaría agradecer en especial el gran esfuerzo que han tenido que hacer mis padres para hacer posible esta oportunidad.

También me gustaría mencionar a mis amigos "de toda la vida" por poder sobrellevar el no verme durante semanas cuando la universidad se hacía realmente complicada.
Aprovecho mencionar a los amigos que he conocido en esta etapa, ya que gracias a ellos la universidad se ha hecho una experiencia inolvidable.

Por último agradecer a mi tutor toda la ayuda prestada y sus sabios consejos cuando no sabía cómo continuar con mi proyecto. También me gustaría pedirle perdón por no haber cumplido mi palabra de enviar mi trabajo todas las semanas.

Por estas razones ¡¡GRACIAS A TODOS!!

Índice General

Índice de figuras

1. Introducción

Las anotaciones son una forman de añadir *metadatos* al código fuente. Estos datos están disponible para la aplicación en tiempo de ejecución, también pueden usarse por el IDE durante la programación, como por ejemplo la anotación *@override*. Las anotaciones pueden añadirse a los elementos del programa tales como: Clases, Métodos, Atributos, Paquetes

Las anotaciones para el lenguaje Java se incluyeron en septiembre de 2004, a partir de la versión 1.5 de *JDK [0]*, aunque se presentaron en el *Java Community Process* en 2002 como especificación de *JSR-175*. Las anotaciones surgieron por la necesidad de eliminar el código repetitivo de muchas de las APIs utilizadas en el momento.

El procesamiento de esta nueva funcionalidad se realiza cuando se compila el código fuente. El compilador de Java almacena los metadatos contenidos de la anotación en los ficheros de clases. Esta información es usada por la *Máquina Virtual de Java* u otros programas que lean la información de los metadatos, como por ejemplo los procesadores de anotaciones.

La ventaja que tienen las anotaciones con respecto a otro tipo de etiquetas es que las anotaciones son completamente accesibles al programador. Mientras que las otras etiquetas necesitan herramientas externas como por ejemplo *XDoclet*.
Potenciando esta característica se ha podido realizar este trabajo de fin de grado.

1.1. Motivación

Antes de nada comenzaremos explicando que es la librería Swing. Swing es una biblioteca gráfica para el lenguaje de programación Java. Esta librería incluye elementos para componer interfaces gráficas. Estos elementos son botones, cajas de textos, desplegables…
Sus principales ventajas son:

- Las interfaces gráficas son independientes de la plataforma.
- Personalizable, los programadores pueden personalizar las interfaces generadas con Swing.

Dado que la creación de las interfaces gráficas con swing son lentas, costosas y pueden dar problemas al diseñarlas, se ha buscado una solución. Esta solución es la generación de código por medio de los metadatos de las anotaciones. Estos metadatos se procesan en el motor de anotaciones y se genera código compilable en java con las librerías de swing, con las características obtenidas del programa y las anotaciones.

Con esta operativa se consigue que de una sola programación y con unas sencillas anotaciones se genere una interfaz gráfica de usuario.

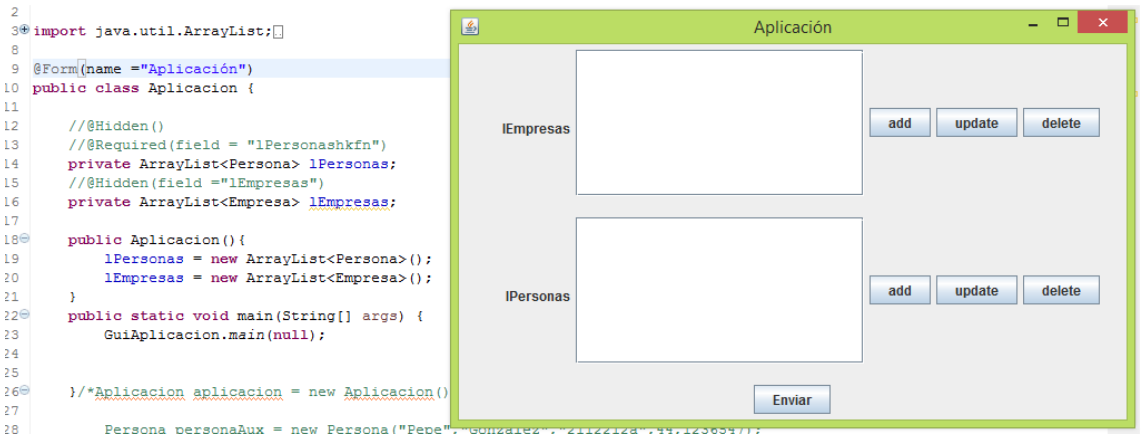

Figura 1: Ejemplo interfaz gráfica 1

Como se puede apreciar en la figura 1, la interfaz gráfica está compuesta por los datos leídos de los atributos de la clase, estos atributos son lPresonas y lEmpresas, y la información básica proporcionada por la anotación @Form.

1.2.Objetivos y Organización del documento

El objetivo es crear un conjunto de anotaciones que interactuando con un procesador de anotaciones, sea capaz de describir una interfaz gráfica con los elementos anotados. Tanto el conjunto de anotaciones como el procesador se describirán en este documento. Primero se mostrarán los trabajos existentes de generación de interfaces gráficas y que cual ha sido la motivación para crear este proyecto. A continuación se mostrará la información técnica necesaria para poder comprender el trabajo realizado. En el siguiente apartado se explicará cual ha sido el diseño y como se ha implementado toda la funcionalidad. Para finalizar se mostrará un ejemplo y las conclusiones.

2. Trabajo relacionado

El campo del diseño de anotaciones y procesadores en Java está en auge. Poco a poco van apareciendo nuevas formas de utilización de las anotaciones. Por poner algunos ejemplos tenemos librerías basadas en anotaciones y reflexión para crear pruebas unitarias, o se utiliza una extensión de Java para expandir el ámbito de aplicación de las anotaciones (a bloques de código y expresiones), o por medio de un conjunto de anotaciones y un procesador de anotaciones se genera código.
En este último punto nos centraremos, ya que la aplicación descrita en este documento pertenece a este tipo.

Dado que el trabajo aquí explicado tiene como funcionalidad la generación de interfaces gráficas, veremos que aplicaciones existen para realizar esta operativa.

- **Semantic Annotation for Java** [1], esta funcionalidad ha sido creada por Douglas Lyon. Esta aplicación permite crear interfaces gráficas sencillas por medio de anotaciones. Estas anotaciones se utilizan sobre atributos y tienen como parámetros las propiedades del campo anotado. Anotando varios atributos conseguimos generar una interfaz gráfica sencilla. A continuación veremos un ejemplo.
    - Método para generar un control gráfico para un campo boolean.

```
@BooleanRange(
            getValue = true,
            getName = "isVisible"
)
public void setVisible(boolan visible){
      this.visible = visible;
}
```

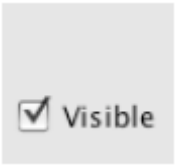

Figura 2: Ejemplo Generación de interfaz gráfica 1

Como se pueden apreciar en la *Figura 2*, en la anotación @BooleanRange hay que definir si el checkBox está a verdadero/falso y el nombre para mostrar.

- Método para generar un control gráfico para un campo Float.

```
@FloatRange(
            getValue = 10,
            getMin = 1,
            getMax = 100,
            getName = "x",
            getIncrement = 0.01f
)
public void setX(float x) {
      this.x = x;
}
```

Figura 3: Ejemplo Generación de interfaz gráfica 2

Como se puede apreciar en esta ocasión, para generar la interfaz gráfica es necesario completar los datos de los valores máximo y mínimo, así como el tipo de incremento y el nombre del campo.

- Metawidget [2], esta aplicación permite generar componentes dinámicamente dentro de interfaces gráficas ya creadas. Esta aplicación crea un nuevo componente en la interfaz gráfica, y este componente se rellena con los datos que lee de una clase dada. Para leer los datos de la clase utiliza los métodos get y set. A continuación se puede ver un ejemplo de su utilización.

```
public class Person{
          private String name;
          private int age;
          private boolean retired;
}
```

```
SwingMetawidget metawidget = new SwingMetawidget();
metagidget.setToInsoect(person);
JFrame frame = new JFrame("Ejemplo Metawidget")
Frame.getContentPane().add(metawidget);
```

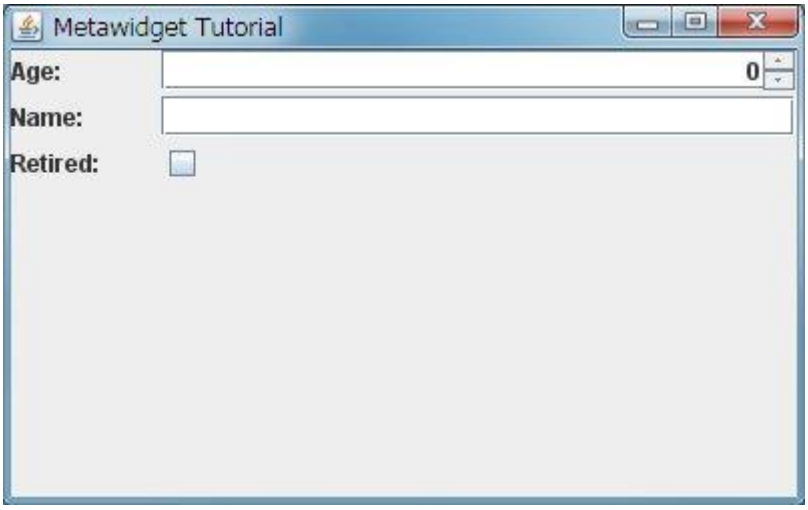

Figura 4: Ejemplo Generación de interfaz gráfica 3

Como se puede ver en el ejemplo, en este caso se lee la información de la clase person y se compone la interfaz gráfica, pero el programador no tiene ningún control sobre ella.

- Object 2 Gui [3], esta aplicación permite generar interfaces gráficas por medio de anotaciones de Java. En este caso toda configuración de la interfaz recae sobre las anotaciones. A continuación se puede ver un ejemplo.

```
@GUI_CLASS(type=GUI_CLASS.Type.BoxLayout,
BoxLayout_property=GUI_CLASS.Type_BoxLayout.Y)
public class Test
{
        public enum Rank { DEUCE, THREE, FOUR, FIVE, SIX, SEVEN,
EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE };

        @GUI_FIELD_TYPE(type=GUI_FIELD_TYPE.Type.LIST)
        private Rank Enum_2 = Rank.TEN;
```

```
        @GUI_FIELD_TYPE(type=GUI_FIELD_TYPE.Type.TEXTFIELD)
        private float Valeur1 = 12.5f;

        @GUI_FIELD_TYPE(type=GUI_FIELD_TYPE.Type.SLIDER, min=0,
max=10, divider=1000)
        private float Valeur2 = 0f;
}
```

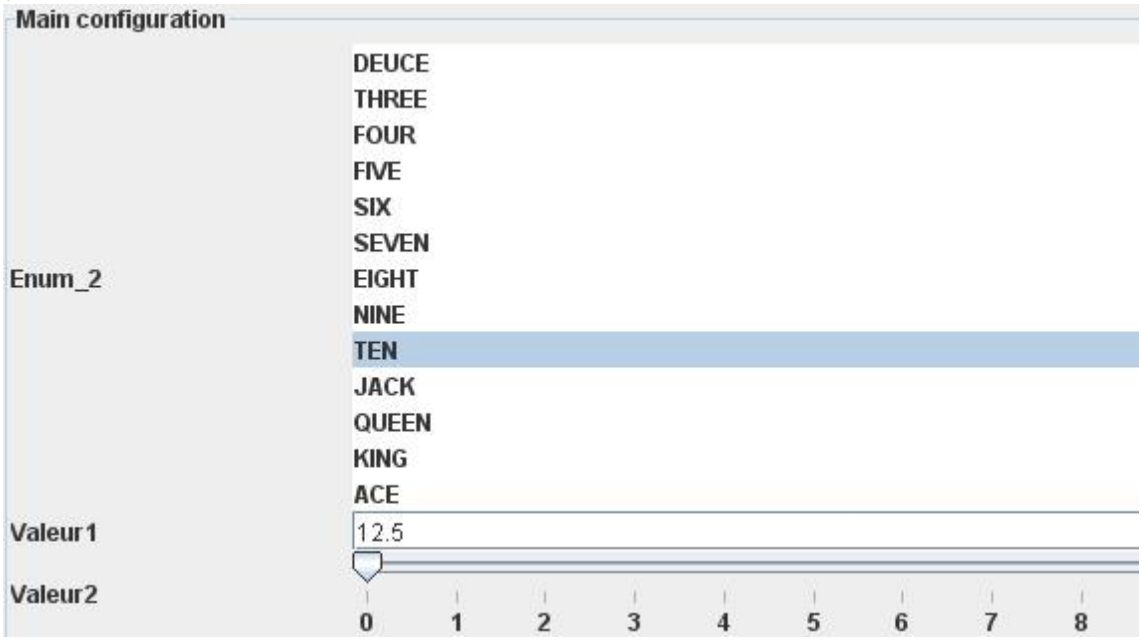

Figura 5: Ejemplo Generación de interfaz gráfica 4

Como se puede ver en este ejemplo, la carga de trabajo sobre las anotaciones es muy grande. Toda la generación de interfaces gráficas está en los parámetros que se rellenan de la anotación.

- Model-Driven GUI Generation [4], creado por Javier Paniza. Esta aplicación permite crear interfaces gráficas por medio de anotaciones en el nivel de clase. Tan solo se anota la clase de la que se quiere obtener el formulario. A continuación veremos un ejemplo.

```
@Entity
public class Customer {

        private int number;
        private String name;
}
```

Figura 6: Ejemplo Generación de interfaz gráfica 5

Como se puede ver esta interfaz coge los valores de los atributos y genera las etiquetas y los campos de texto.

Tras ver todos los trabajos relacionados que existen sobre el área de la generación de código para interfaces de usuario, se vieron una serie de carencias que se podían mejorar. Por esa razón se comenzó a trabajar en el ***APT Processor***. Esta aplicación tiene las ideas de los trabajos mostrados anteriormente y algunas mejoras que pueden hacer que su utilización predomine sobre el resto,
Estas mejoras son:

- Realizas anotaciones ligeras y sencillas. Con esta mejora se quiere evitar tener que ser un experto en la materia para poder utilizar esta aplicación, como ocurre para los casos de Semantic Annotation for Java y Object 2 Gui. Por esta razón, como se verá en el apartado 4.2 Anotaciones, el conjunto de anotaciones que utiliza el ***APT Processor*** es bastante más sencillo.
- Introducir la posibilidad de personalizar los campos de la interfaz gráfica de usuario. Con esta mejora se quiere dar la posibilidad de que el programador pueda "personalizar" el color o los nombres de las ventanas generadas. Con esta opción se evitaría que ocurriera como en el caso de Model-Driven GUI Generation o Metawidget, que genera toda la interfaz pero no permite al programador modificar nada dentro de la ventana.

3. Estudio de las tecnologías

En este apartado se estudiarán las distintas tecnologías, conceptos y herramientas que se han utilizado para el desarrollo del ***APT Processor*** y que serán mencionados con frecuencia en el documento.

3.1.Anotaciones en Java

Como se ha comentado anteriormente, las anotaciones son una forma de añadir metadatos al código fuente. Esta tecnología ha facilitado la labor de los programadores y ha permitido que muchos de los *frameworks* actuales *Spring*, *JUnit*, *Hibernate*, etc, mantengan separado el comportamiento del sistema con sus tareas, dado que utilizan las anotaciones y los metadatos para conseguir su fin. Con estas acciones se mejora la limpieza del código y se facilita su legibilidad.

Para poder interpretar estos metadatos, se suele utilizar un procesador de anotaciones. Este procesador transforma los metadatos recogidos por medio de las anotaciones en la funcionalidad requerida o especificada por el programador. Estos procesadores pueden ser muy variados.

En este caso concreto, utilizando la información de las anotaciones y generando código a través del procesador de las anotaciones, podemos conseguir que se generen de manera dinámica las interfaces gráficas (utilizando *swing*) que el usuario defina por medio de **@**(Nombre de la anotación)

3.1.1. Miembros

Los campos de una anotación se declaran de una manera especial: son métodos sin argumentos, que no pueden lanzar excepciones y cuyo tipo de retorno se indica en el tipo del campo. Para asignarles valores por defecto se utiliza la palabra clave default y a continuación se asigna el valor.
Una anotación puede estar compuesta por varios miembros o ninguno y no podrá contener más miembros de los declarados de forma explícita. Las anotaciones que no contienen ningún miembro son nombradas como *anotaciones de marcado.*
Los miembros que componen una anotación pueden ser de los siguientes tipos de datos.

- Tipo de datos primitivo, como pueden ser: Short, Int, Long, Float, Double, Char, Boolean.
- Tipo de datos que derivan de clases, como son: String, Class, Invocaciones parametrizadas de Class
- Otras anotaciones
- Colecciones, pero solo si están compuestas por tipos de datos primitivos, como por ejemplo: Arrays y Listas

A continuación se muestra un ejemplo de definición de una anotación utilizada por el ***APT Processor***:

```
public @interface Form {
	String name() default "";
	String background() default "";
}
```

Para diseñar el conjunto de anotaciones que utiliza el procesador de anotaciones ***APT Processor***, se han utilizado anotaciones con tipo de datos primitivos y tipo de datos que derivan de clases.

3.1.2. Elementos a anotar

El objetivo principal de las anotaciones en marcar un elemento de un programa en Java.Al marcarlo, obtenemos metadatos del elemento, muy útiles a la hora de realizar el proceso del motor de anotaciones. Por esta razón es importante conocer los distintos elementos que se pueden marcar. Los elementos sobre los que se pueden utilizar estas anotaciones son: Paquetes, Clases, Interfaces, Enumeraciones, Anotaciones, Constructores, Métodos, Atributos, Variables locales, Bucles.

Como se puede apreciar, prácticamente se puede utilizar la tecnología de las anotaciones en cualquier lugar del código Java. Para la realización del diseño del ***APT Processor*** únicamente utilizaremos las anotaciones sobre clases y atributos.

3.1.3. Uso de anotaciones

Existen dos maneras de utilizar las anotaciones en Java. La primera manera es marcar el elemento por encima, como se puede apreciar en el ejemplo 1.

```
@Form(name ="Aplicación")
public class Aplicacion {

}
```

Y la segunda manera es utilizar la anotación dentro de la definición de la clase, como se puede ver en el ejemplo 2.

```
public @Form(name ="Aplicación") class Aplicacion {

}
```

Generalmente se utiliza la primera manera de anotar, ya que es más comoda para el programador. Podemos ver casos donde se utiliza esta manera de anotar, como por ejemplo en las clases autogeneradas en el entorno de desarrollo de Eclipse.

3.1.4. Procesamiento de anotaciones

Para poder leer las anotaciones descritas en los puntos anteriores, es necesario utilizar la funcionalidad que proporciona Java en el paquete *javax.annotation.processing*. Este paquete contiene las herramientas necesarias

para procesar la información obtenida por las anotaciones en el tiempo de compilación.

Los procesadores de anotaciones utilizan métodos para buscar las anotaciones (@...) en el código fuente. En el caso de que encuentren algún error o warning en la compilación la mostrarán. El ***APT Processor*** utiliza esta técnica para definir al usuario donde puede utilizar cada tipo de anotación.
El método que utiliza el compilador para procesar todas las anotaciones funciona por rondas: en cada ronda se puede requerir a un procesador que procese un subconjunto de anotaciones encontradas en el código fuente y los archivos producidos en una ronda anterior.
Con este método e evita que en caso de que haya una generación de nuevas anotaciones se queden sin procesar.

La sintaxis utilizada en los procesadores de anotaciones está compuesta por las anotaciones de:

- *SupportedAnnotationTypes,* esta anotación indica los tipos de anotaciones que soporta el procesador. Esta anotación contiene un parámetro, este parámetro indica el nombre del tipo de anotación a procesar o bien un asterisco '*' para indicar que procese todos los tipos de anotaciones.
- *SupportedSourceVersion*, esta anotación indica el tipo de versión que soporta el procesador. Para nuestro caso será la versión *Release_6.*
- *Override del método process*, el método process es la función principal del procesador de anotaciones. En el apartado 4.1 se explicará su funcionamiento.

3.2.Nuevas nociones de programación

Esta nueva forma de programación con metadatos abre muchas nuevas posibilidades de programación. Ya que se pueden analizar los datos en tiempo de compilación e interactuar con ellos. De este modo se pueden crear generadores de códigos específicos para una aplicación. Como sería el caso de este trabajo. También se pueden utilizar estos metadatos para conseguir lenguajes orientados a atributos o lenguajes de dominio específicos.

4. Diseño e Implementación

En este apartado se estudiará el diseño y la implementación del ***APT Processor***.
Así como el diseño y la implementación específica de cada uno de los componentes.

4.1.Procesador de Anotaciones

En este apartado de explicará el funcionamiento del método principal de un procesador de anotaciones, el método P**rocess**. A continuación se muestra su estructura.

```
@Override
public boolean process(Set<? extends TypeElement> elements,
                       RoundEnvironment env)
```

- El primer parámetro es el conjunto de anotaciones que se evalúan en la ronda actual.
- El segundo parámetro objeto con los elementos anotados por las anotaciones en la ronda actual.
- El retorno, este método retorna un valor booleano, en el caso de que este valor sea *true*, quiere decir que el procesador reclama las anotaciones que ha procesado y ningún otro procesador se ejecutará sobre estas anotaciones. En el caso contrario, que el valor sea *false*, el procesador no reclama ninguna anotación y por tanto distintos procesadores analizarán la información.

Jugando con estas posibilidades se puede conseguir que una misma anotación tenga características distintas o se realicen acciones distintas dependiendo del procesador que las procese.

4.2.Anotaciones

En este apartado se explicarán las anotaciones que utilizan el ***APT Processor*** y el funcionamiento de las mismas.
La funcionalidad principal recae sobre la anotación **form,** que es la encargada de generar el código en swing, el resto de anotaciones solo tiene como objetivo modificar el código generado por la anotación **form**, para personalizarlo a petición del usuario.

4.2.1. Form

Esta anotación es la anotación principal, soporta todo el peso de la generación de código.

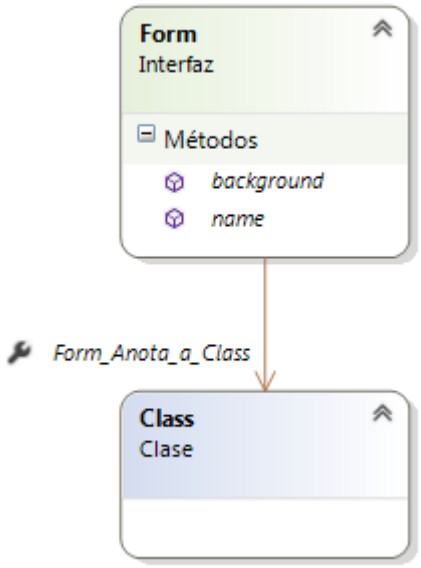

Figura 7: Diagrama anotaciones sobre clases

Esta anotación solo puede utilizarse en clases.
El procesador de código al encontrar una clase marcada con esta anotación, realiza una lectura de todos los elementos de la clase. En esta lectura se identifican en elementos en dos tipos, los necesarios para la interfaz gráfica y los que no. Con el grupo necesario para la interfaz gráfica, se componen las ventanas. Para ello se lee el tipo de dato que es ese elemento y se busca el componente de swing que más se adecue. A continuación se muestran varios ejemplos.

Si el elemento leído es un atributo de tipo *int*, (`private int codigoPostal;`), se implementará un componente de swing que soporte numérico, como es el caso de un *TextBox*, este campo tiene una serie de validaciones para no permitir caracteres que no sean numéricos.

Si el elemento leído es un atributo de tipo *boolean*, (`private bolean apto;`), se implementará el componente checkBox, ya que este componente es el que más se adecua a las necesidades del campo.

Si el elemento leído es un atributo de tipo *class*, (`private Empresa empresa;`), se implementará un componente del tipo *Lista*, ya que como desconocemos que tipo de datos almacenará la clase, utilizamos un componente muy versátil.

Como se puede ver en estos tres ejemplos, el ***APT Processor*** se adecua a cada tipo de dato para obtener la interfaz gráfica más óptima.

Esta anotación contiene parámetros, que se utilizarán para personalizar los formularios generados, no son obligatorios. Los parámetros son:
Name: Este parámetro define el nombre de la ventana del formulario con una cadena de texto.

Background: Este parámetro modifica el color de fondo del formulario, hay que escribir el nombre de un color en inglés para que funcione correctamente.
Un ejemplo de la utilización de esta anotación:

```
@Form(name ="Aplicación", background = "Blue")
public class Aplicacion {
```

Para el resto de anotaciones el diagrama será el siguiente.

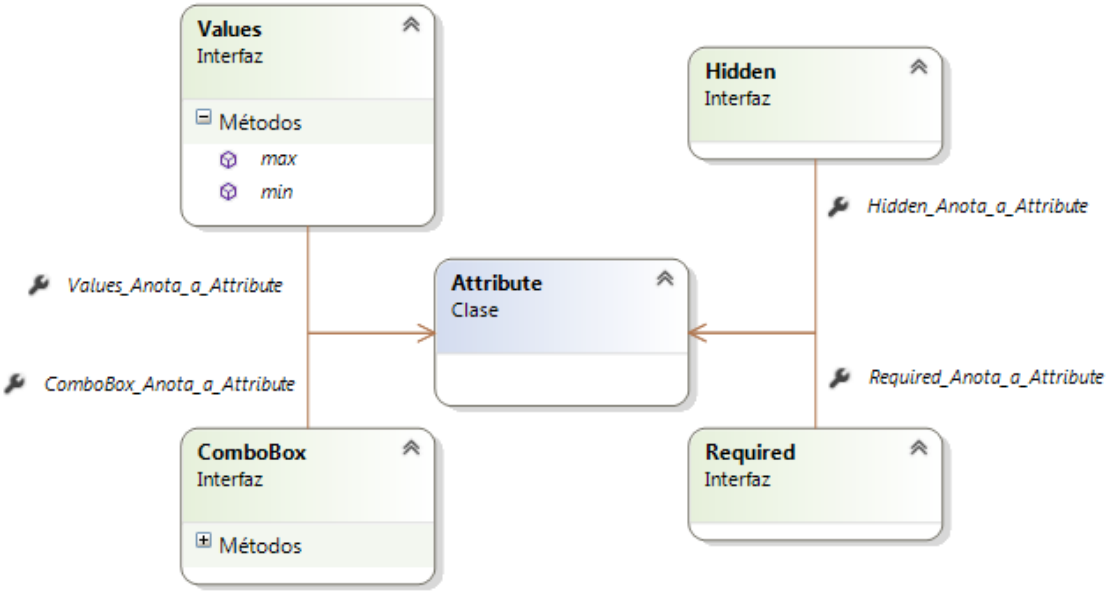

Figura 8: Diagrama anotaciones sobre atributos

4.2.2. Required

En el conjunto de anotaciones, la anotación *required* se utiliza para dar la propiedad de obligatorio a un elemento de la interfaz gráfica. Esto significa que el usuario que utiliza esa interfaz, no podrá dejar ese campo en blanco.
Para utilizar esta anotación el programador deberá utilizar la anotación sobre un atributo de su clase. Como esta anotación no tiene parámetros, la utilización de la anotación será como en el siguiente ejemplo.

```
@Required
private String dni;
```

4.2.3. Hidden

En el conjunto de anotaciones, la anotación *hidden* se utiliza para ocultar un elemento en la interface gráfica. Esto significa que el usuario no verá el campo con la propiedad de oculto.
Para utilizar esta anotación el programador deberá utilizar esta anotación sobre un atributo de su clase. Como esta anotación no contiene parámetros, la utilización de la anotación será como en el siguiente ejemplo.

```
@Hidden
private String nombre = "Oscar";
```

4.2.4. Values

En el conjunto de anotaciones, la anotación *Values* se utiliza para dar la propiedad de valores entre un rango determinado a un elemento de la interfaz. Esto significa que el usuario deberá completar los datos del formulario con un valor que este entre los rangos definidos por el programador. Esta anotación solo se puede utilizar en atributos de tipo *integer*.

Para utilizar esta anotación el programador deberá utilizar esta anotación sobre un atributo de tipo *integer*. Esta anotación necesita parámetros para poder definir los rangos, estos parámetros son: **min**, que es el número mínimo del rango y **max**, que es el número máximo del rango. En caso de no rellenar ningún parámetro se tomará cero como mínimo y 2.147.483.647 para el máximo. A continuación veremos un ejemplo de utilización de la anotación *Values*

```
@Values(min = 2, max = 10)
private int edad;
```

4.2.5. ComboBox

En el conjunto de anotaciones, la anotación *ComboBox* se utiliza para inicializar valores en un determinado elemento de la interfaz. Esto significa que el usuario podrá seleccionar los valores definidos por el programador en un menú desplegable.

Para utilizar esta anotación el programador deberá usarla sobre un atributo de su clase. Esta anotación necesita un parámetro, este parámetro es de tipo String[] y contiene la información de los elementos que aparecerán en el menú desplegable. A continuación se muestra un ejemplo de su utilización.

```
@ComboBox(listado = {"Madrid","Barcelona","Valencia"})
public String[] provincias;
```

4.3.Generador de código

El procesador de anotaciones ***APT Processor***, utiliza el conjunto de anotaciones definido en el apartado 4.2 Anotaciones para generar código swing. Este código generado es una interfaz gráfica de usuario. Se separa en tres tipos JFrames, JPanels, y los controladores para gestionar las interfaces. Estas clases poseen toda la información y la personalización que el programador ha definido. Esta información se compone de la siguiente manera:

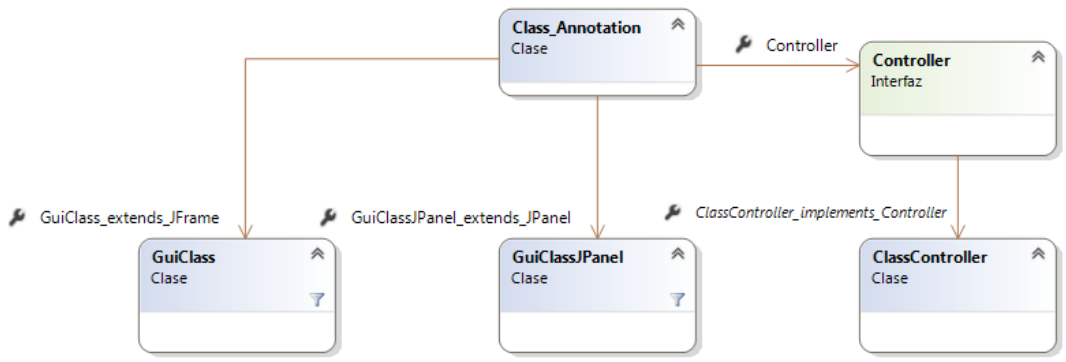

Figura 9: Diagrama de composición de las clases generadas

Todo el código generado sigue el patrón de arquitectura del software *Modelo-Vista-Controlador*[5]. Este tipo de arquitectura permite separar los datos de la lógica del negocio. Para poder conseguir este objetivo se construyen tres componentes, que son:

- Modelo, es la representación de la información con la cual el sistema opera. En el caso de este proyecto son las clases generadas por el desarrollador.
- Vista, es el encargado de presentar los datos del modelo en un formato adecuado, en el caso del ***APT Processor*** este formato son las interfaces gráficas de usuario.
- Controlador, es el encargado de responder a los eventos que unen las vistas con el modelo, en este caso las clases encargadas de cumplir esta funcionalidad son las clases que implementan la interfaz Controllers.

A continuación se muestra un diagrama de interacción del usuario con el sistema.

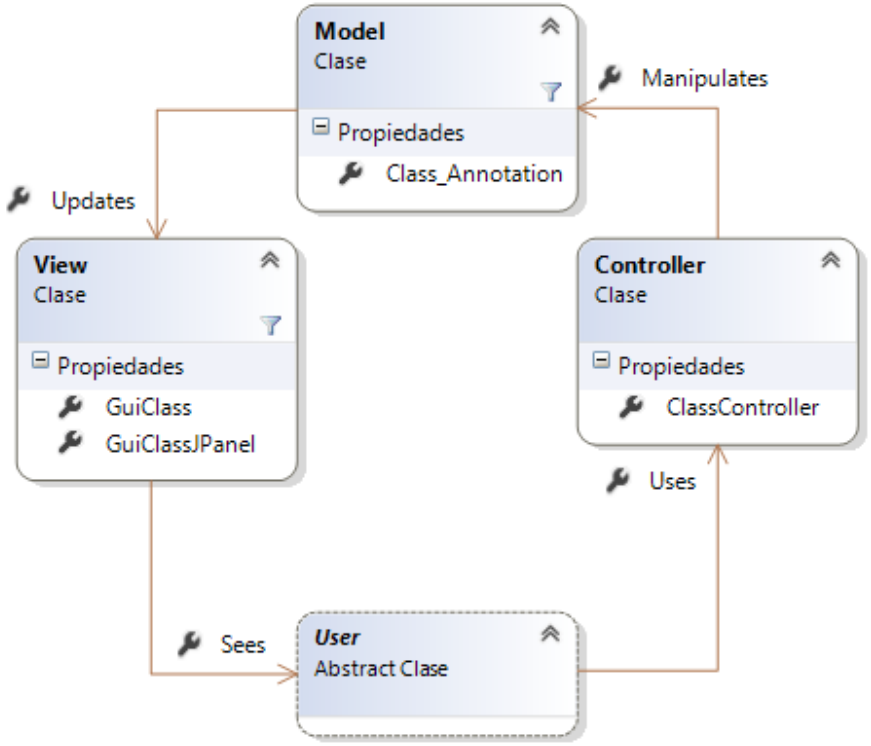

Figura 10: Diagrama de utilización del usuario

Como se puede ver en el diagrama, el modelo actualiza la información de las vistas, las vistas presentan al usuario la información, el usuario utiliza esa información para modificar los datos y el controlador manipula los datos que almacena el modelo.

Una vez explicado cuales son los puntos del generador de anotaciones y como se va a estructurar se procede a describir de manera detallada la funcionalidad y la estructura que tiene los tres tipos de generación de código.

4.3.1. Generador de JFrames

El generador de JFrames pertenece a la parte de la Vista en la arquitectura Modelo-Vista-Controlador. Se comenzará explicando brevemente que es un JFrame. JFrame es una clase utilizada en la biblioteca gráfica de Swing. Se utiliza para generar ventanas sobre las cuales se pueden añadir distintos componentes (botones, etiquetas, campos de texto,…) con los que el usuario podrá interactuar o no. Se diferencia de un JPanel en que el JFrame posee propiedades propias de una ventana, como puede ser maximizar, minimizar, cerrar y poder moverla.

Una vez explicado que es un JFrame procedemos al diseño e implementación. Tras la obtención de la información requerida de las anotaciones, nombres de atributos, métodos, tipo de datos… se comienza a generar el código de una manera ordenada.
Se comienza generando la información del paquete donde estarán todos nuestro JFrames, ese paquete será **gui** (graphical user interface)

El siguiente código a generar será el de los imports necesarios para el formulario. Cabe destacar la manera de generar la importación del origen de datos, ya que se obtiene por el origen de la anotación @form. Gracias a este método de obtención del origen podemos utilizar los constructores de las clases.

A continuación de declaran la cabecera de la clase, que por convenio se llamará GUI + el nombre de la clase anotada y extenderá de Jframe, Ej.

Clase:

```
@Form(name ="Aplicación", background = "Blue")
public class Aplicacion {  }
```

JFrame:

```
public class GuiAplicacion extends JFrame{ }
```

Después de generar las cabeceras de las clases se declaran los atributos necesarios para la clase, estos atributos son:

- La instancia del formulario, para poder utilizar en los controladores
- La instancia del panel que rellena el JFrame.

El siguiente paso es la generación de los constructores, en estos constructores se inicializan las variables recogidas por las anotaciones para llamar al constructor del JPanel.
Existen dos tipos de constructores, un sin parámetros que contiene la inicialización y llama al constructor "completo" que utiliza todos los datos pasados por referencia para llamar al constructor de la clase JPanel, además realiza las validaciones de rango, y obligatoriedad de los campos.

El siguiente fragmento de código que se genera son los métodos para visualizar u ocultar el formulario, estos métodos son utilizados por los controladores.

A continuación se declaran los métodos getters y setters.

Y por último se genera un método main, que se utiliza para las pruebas unitarias de los formularios.
A continuación se muestra un ejemplo de la generación de JFrame de la clase *Aplicación.*

```
public class GuiAplicacion extends JFrame{
	private  static final GuiAplicacion instanceGuiAplicacion =
new GuiAplicacion();
	private GuiAplicacionJPanel form;
	public GuiAplicacion(){
		ArrayList<String> labels = new ArrayList<String>();
		ArrayList<String> descs = new ArrayList<String>();
		int width = 15;
		labels.add("lEmpresas");
		labels.add("lPersonas");
		GuiAplicacionA("Aplicación",labels,width,descs);
	}
	public void GuiAplicacionA(String title,ArrayList<String>
labels, int width, ArrayList<String> tips) {
		form = new GuiAplicacionJPanel(labels, width, tips);
		setTitle(title);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		getContentPane().add(form, BorderLayout.NORTH);
	JButton submit = new JButton("Enviar");
		submit.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
			HideGUI();
			}
		});
		JPanel p = new JPanel();
		p.add(submit);
		getContentPane().add(p, BorderLayout.SOUTH);
		pack();
		setVisible(true);
	}
	//Funciones auxiliares
	//Main
}
```

Como se puede apreciar, el código generado tiene dos constructores, el primer constructor (`GuiAplicacion`) es el encargado de inicializar los valores de las listas de atributos. El segundo constructor (`GuiAplicacionA`) es el encargado de crear los paneles llamando al constructor de la clase JPanel, configurar el Jframe y el botón de acción.

4.3.2. Generador de JPanels

El generador de JPanels pertenece a la parte de la Vista en la arquitectura Modelo-Vista-Controlador. Se comenzará explicando brevemente que es un JPanel. Un JPanel es una clase utilizada en la biblioteca gráfica de Swing. Comparte las

propiedades principales de un JFrame, pero solo puede estar dentro de un JFrame, ya que por sí sola no es una ventana.
Al igual que en el caso de los JFrames, comenzamos a explicar la generación de código de manera ordenada.
Se comienza generando el código de los imports necesarios para la clase generada. En este caso se necesitan las librerías de swing y los action listener. Cabe destacar como en el caso anterior, la importación del fichero fuente donde se realiza la anotación *@form* y la importación del paquete con los controladores.
Una de las librerías más importes que se utilizan en esta clase, es la librería de *java.lang.reflect*. Esta librería permite "reflejar" y examinar los componentes en tiempo de ejecución. Por medio de este método podemos rellenar los datos de los objetos generados así como obtener sus valores. Gracias a esta librería es posible generar el código que interactúan entre distintas anotaciones y de las que no conocemos ningún dato.

A continuación de declaran la cabecera de la clase, que por convenio se llamará GUI + el nombre de la clase anotada y en este caso extenderá de JPanel, Ej.
Clase:

```
@Form(name ="Aplicación", background = "Blue")
public class Aplicacion {  }
```

JFrame:

```
public class GuiAplicacionPanel extends JPanel{ }
```

El siguiente paso es la declaración de los atributos. En este caso se declaran todos los componentes de swing, JLabels, JButtons, JList, JTextField…

El siguiente paso en la generación de código es un método que rellena con datos todos los atributos. Se decidió utilizar este método de inicialización para facilitar la modificación por parte del usuario, ya que fácilmente podrá modificar este método para cambiar la inicialización.

El próximo paso es la generación del constructor. Este constructor va generando paneles que rellena con los datos de un atributo de la clase anotada con *@form*. Con toda esta información se genera un panel compuesto de subpaneles y estos subpaneles contienen los componentes.

A continuación se generan los métodos de get y set de modificación de la clase principal. Esto se consigue por medio de la reflexión de Java.

Por último se generan las funciones de validación de datos, estas funciones pueden aparecer o no. Ya que depende directamente de la anotación *@Value.*

A continuación se muestra un ejemplo de la generación de JPanel para la clase *Aplicación.*

```
public class GuiAplicacionJPanel extends JPanel{
```

```
	private JPanel p;
	private JLabel lEmpresasLabel = new JLabel("lEmpresas",
JLabel.RIGHT);
	private JList<?> lEmpresas = new JList<Object>();
	private JScrollPane lEmpresasScroll = new
JScrollPane(lEmpresas);
	private DefaultListModel lEmpresasModel = new
DefaultListModel<String>();

	public ArrayList<String> getInfoForm(){
		ArrayList <String> lAtributos = new
ArrayList<String>();
		return lAtributos;
	}

	public GuiAplicacionJPanel(ArrayList<String> labels, int
width, ArrayList<String> tips) {
		super(new BorderLayout());
		JPanel labelPanel = new JPanel(new
GridLayout(labels.size(), 1));
		JPanel fieldPanel = new JPanel(new
GridLayout(labels.size(), 1));

		//Se define el tamaño de los paneles de Etiquetas y
Componentes
		labelPanel.setPreferredSize(new Dimension(100, 300));
		fieldPanel.setPreferredSize(new Dimension(500, 300));

		add(labelPanel, BorderLayout.WEST);
		add(fieldPanel, BorderLayout.CENTER);

		//lEmpresa
		lEmpresasLabel.setLabelFor(lEmpresas);
		lEmpresasScroll.setPreferredSize(new Dimension(200,
35));		labelPanel.add(lEmpresasLabel);

		p = new JPanel(new FlowLayout(FlowLayout.LEFT));
		lEmpresas.setSelectedIndex(1);
		lEmpresas.setModel(lEmpresasModel);
		p.add(lEmpresasScroll);
		fieldPanel.add(p);
		//Se añade la funcionalidad de añadir nueva lEmpresas
		JButton addlEmpresas = new JButton("add");
		addlEmpresas.addActionListener(new ActionListener(){
			public void actionPerformed(ActionEvent e){
				//Controller
				EmpresaController EmpresaC = new
EmpresaController();
				Object o = EmpresaC.add();
				if(o != null)
					lEmpresasModel.addElement(o);
			}
		});
		p.add(addlEmpresas);

		//Se añade la funcionalidad de modificar lEmpresas
		JButton updlEmpresas = new JButton("update");
		updlEmpresas.addActionListener(new ActionListener(){
			public void actionPerformed(ActionEvent e){
				//Controller
```

```
                        EmpresaController EmpresaC = new
EmpresaController();
                        if(lEmpresas.getSelectedValue() != null)

      EmpresaC.update(lEmpresas.getSelectedValue());
                  }
            });
            p.add(updlEmpresas);

            //Se añade la funcionalidad de eliminar una lEmpresas
            JButton dellEmpresas = new JButton("delete");
            dellEmpresas.addActionListener(new ActionListener(){
                  public void actionPerformed(ActionEvent e){
                        //Controller
                        if(lEmpresas.getSelectedValue() !=  null)

      lEmpresasModel.removeElement(lEmpresas.getSelectedValue());
                  }
            });
            p.add(dellEmpresas);
      }
      public Aplicacion getInstanceClassAplicacion() {
            Aplicacion ret = new Aplicacion();
            Class<?> ce = ret.getClass();
            try {
            } catch (Exception e){
                  e.printStackTrace();
            }
            return ret;
      }
      public void setInstanceClassAplicacion(Object o) {
            Aplicacion ret = (Aplicacion) o;
            Class<?> ce = ret.getClass();
            try {
            } catch (Exception e){
                  e.printStackTrace();
            }
      }
      //Getter y setter de obtención de instancias de la clase
}
```

Como se puede apreciar, el código generado inicializa todos los elementos que necesita como atributos de la clase. En el constructor se crean dos listas de JPanels para almacenar las etiquetas a la izquierda y los componentes editables a la derecha. Tras inicializar estas listas y definir los tamaños de los paneles se comienzan a utilizar los atributos definidos para crear los paneles. Este código generado también tiene dos métodos get y set, que se utilizan para actualizar los datos en las vistas.

4.3.3. Generador de Controllers

El generador de Controllers pertenece a la parte del Controlador en la arquitectura Modelo-Vista-Controlador. Se comenzará explicando que funcionalidad tiene este generador.

Para poder interactuar y dar cohesión a las ventanas generadas, se han implementado controladores. Esto quiere decir que se han creado clases en java que interactúan directamente entre las ventanas y las clases para dar funcionalidad.

Para realizar esta implementación se ha creado una *interfaz* llamada *Controller* que contiene los métodos que son necesarios de completar para cada una de las clases anotadas. Los métodos a completar son add y update ya que se han definido como las acciones más básicas que el usuario puede realizar.

Una vez explicado brevemente que es una interfaz, se comienza a explicar la generación de código.

Como en los casos anteriores se comienza generando el paquete al que pertenece. En este caso el paquete es **controllers**.

A continuación se generar los imports necesarios. Se realiza el import del paquete que contiene las interfaces gráficas. Se necesita para poder interactuar con ellas. Y se realiza el import del paquete donde se encuentran las clases anotadas por la anotación *@form*.
El siguiente paso es la generación de los métodos de la interfaz. Aquí analizaremos cada función para explicar cómo se comporta.

- Método Add: Este método devuelve un Objeto, este objeto es una instancia a la clase a la que hace referencia el formulario. Para conseguir esta funcionalidad el controlador realiza una llamada a un método que contiene el formulario que le devuelve la instancia del objeto. Después muestra el siguiente formulario y devuelve la instancia para almacenarla y mostrarla en el formulario.
- Método Update: Este método solicita un objeto por parámetro, este objeto es una instancia a la clase a la que hace referencia el formulario. El primer paso que se realiza es obtener la instancia del objeto que tiene el formulario. Después se modifican los datos de esa instancia con el objeto pasado por referencia y por último se muestra el siguiente formulario.

4.4.Implementación

En este apartado se van a explicar las características que tiene el conjunto de anotaciones.
Se comenzará explicando las características propias de las anotaciones sobre atributos y posteriormente las características de las anotaciones sobre clases.

4.4.1. Anotaciones sobre atributos

Como se ha explicado en el apartado 4.2, las anotaciones poseen características únicas. A continuación se mostrarán los dos tipos de características que existen.

El primer tipo de características son las anotaciones que solo se pueden utilizar en un determinado tipo de atributo. En este caso se encuentra la anotación *@Values*. Si utilizamos esta anotación en un atributo que no es de tipo *int* aparecerá un error de compilación tal y como se muestra en la siguiente figura.

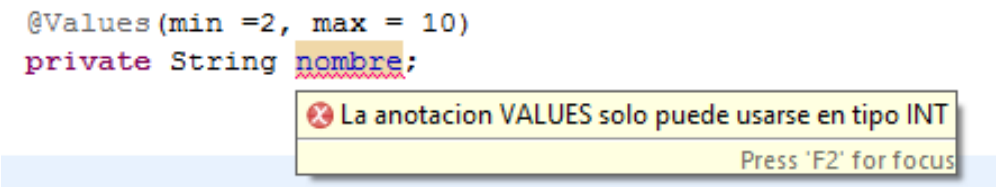

Figura 11: Ejemplo error de anotación, anotación no compatible con el atributo.

El segundo tipo de características son aquellas anotaciones que solo pueden utilizarse en atributos. En este caso se encuentran las anotaciones @Hidden, @Required, @Values, @ComboBox. A continuación se muestra un ejemplo del error que aparecerá al compilar.
Utilización de la anotación @Required para una clase. En esta caso cualquiera de las anotaciones *@Hidden, @Values, @ComboBox o @Required* dan este error.

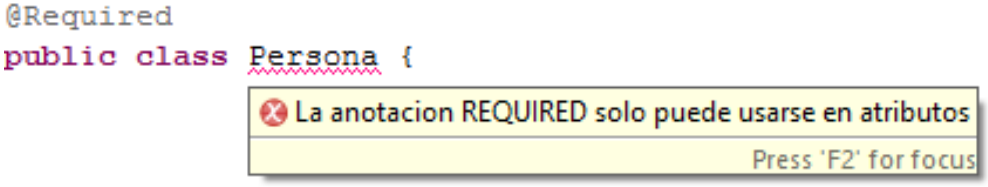

Figura 12: Ejemplo error de anotación, anotación utilizada en una clase

4.4.2. Anotaciones sobre clases

Las anotaciones sobre clases poseen algunas características únicas. A continuación se explicarán cuáles son estas características.

En el conjunto de anotaciones ***APT Processor***, solo se utiliza una anotación sobre clase. Esta anotación es *@Form*, que posee la característica de la personalización de las interfaces gráficas. Esta personalización es posible gracias a los parámetros de entrada de los que dispone la anotación, como se ha explicado anteriormente, estos parámetros son *name* y *background.* Con ellos se puede modificar el nombre y el color que aparece en la ventana.
Para poder utilizar correctamente el parámetro *background*, se necesita que el color se encuentre dentro de la librería de Java *java.awt.Color;*
A continuación se muestra una ilustración donde se muestra el color y el cambio de nombre, así como los parámetros de la anotación.

```
@Form(name ="Nombre de la clase", background = "Cyan")
public class Aplicacion {
      . . .
}
```

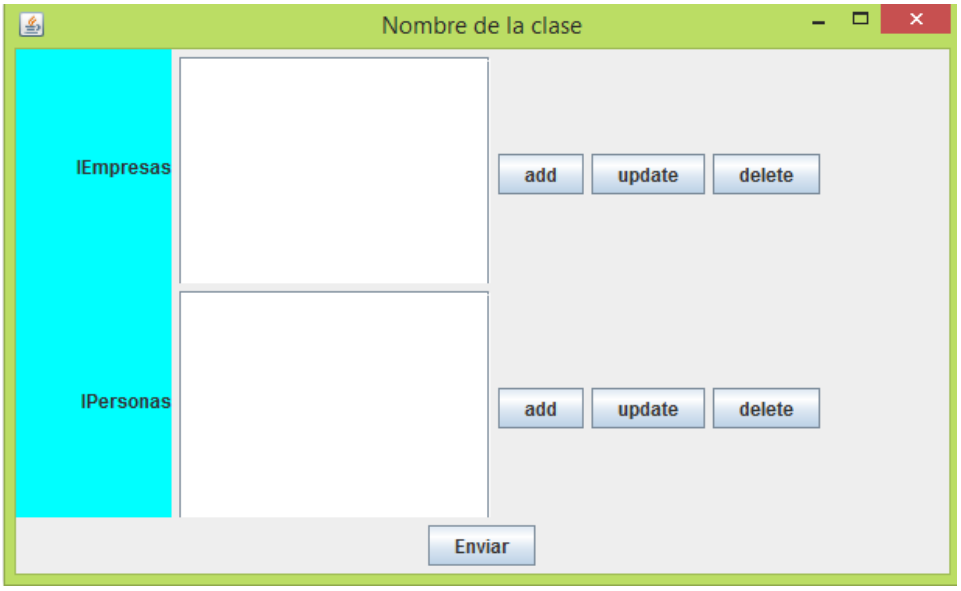

Figura 13: Ejemplo Personalización de interfaces gráficas

5. Ejemplo real

A continuación se mostrará un ejemplo real de cómo utilizar la potencia del ***APT Processor***.

Este proyecto es la gestión de una aplicación de personal. Esta aplicación gestiona personas y empresas y las asigna entre sí. A continuación se muestra un diagrama de clases de la aplicación.

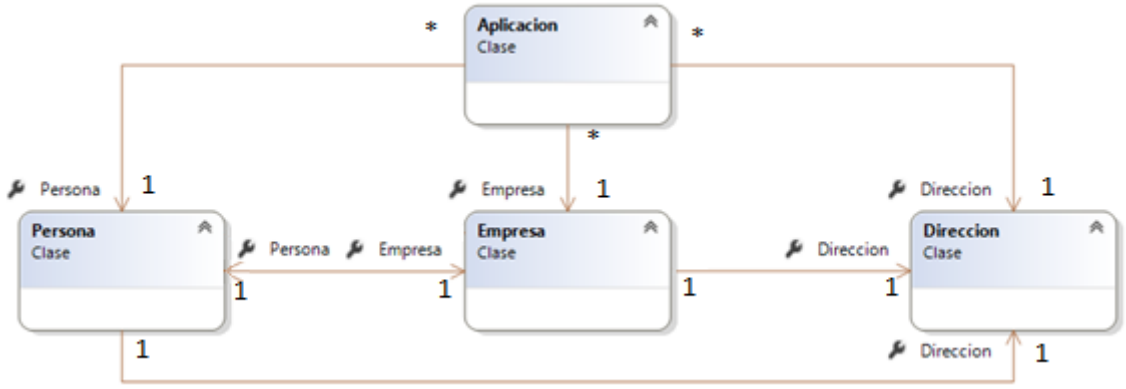

Figura 14: Diagrama de clases de la aplicación

A continuación se describirá los datos que tienen que almacenar cada uno de los datos que gestiona esta aplicación. Se comienza detallando la información almacenada por las personas, Nombre, Apellidos, Dirección, DNI, Edad, Empresa, Teléfono y si la persona está realizando prácticas. A continuación se detalla la información que almacena la Empresa, Nombre, Teléfono, Dirección y Personas. Y por último se explicará la información que almacenan las Direcciones, Calle, Número, Código Postal, Población y Ciudad.

Por esta razón se crearán los formularios para Aplicación, Personas, Empresas y Dirección.

A continuación se mostrará la interfaz gráfica de usuario generada para la clase *Aplicación* y un diagrama de cómo se compone esta información desde el generador de anotaciones ***APT Processor***.

El primer paso será mostrar el código que tiene la clase *Aplicación*.

```
@Form(name ="Aplicación")
public class Aplicacion {
	private ArrayList<Persona> lPersonas;
	private ArrayList<Empresa> lEmpresas;

	public Aplicacion(){
		lPersonas = new ArrayList<Persona>();
		lEmpresas = new ArrayList<Empresa>();
	}
	//main
}
```

Tras utilizar ***APT Processor*** se obtendrá la siguiente interfaz gráfica de usuario. Como se puede apreciar los componentes que tiene la interfaz gráfica son los mismos que los atributos de la clase.

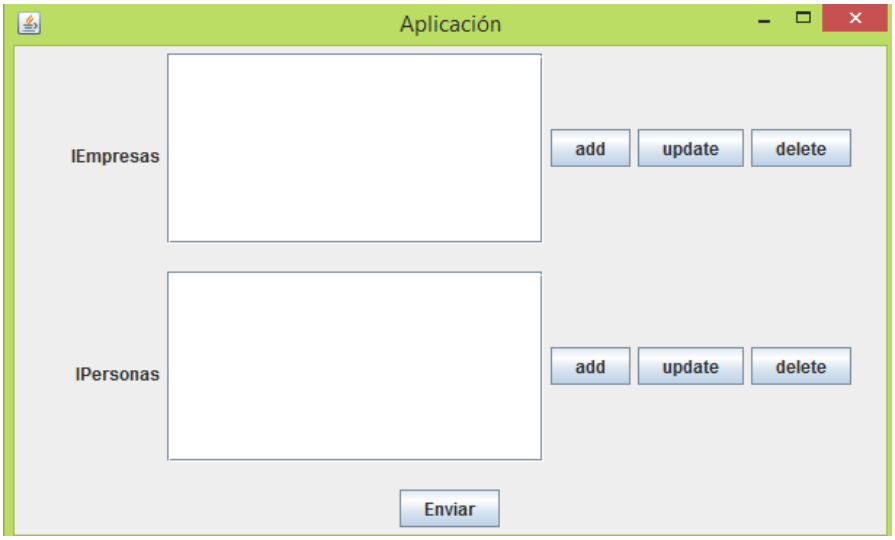

Figura 15: Ejemplo Real, interfaz gráfica de usuario de la clase Aplicación

Como se puede apreciar en la Figura 16, la generación de código que hay detrás de la interfaz gráfica se compone de la siguiente manera. La clase Aplicación genera las clases GuiAplicacion, GuiAplicacionJPanel y AplicacionController al utilizar el conjunto de anotaciones. Esta generación se cumple para todos los casos.

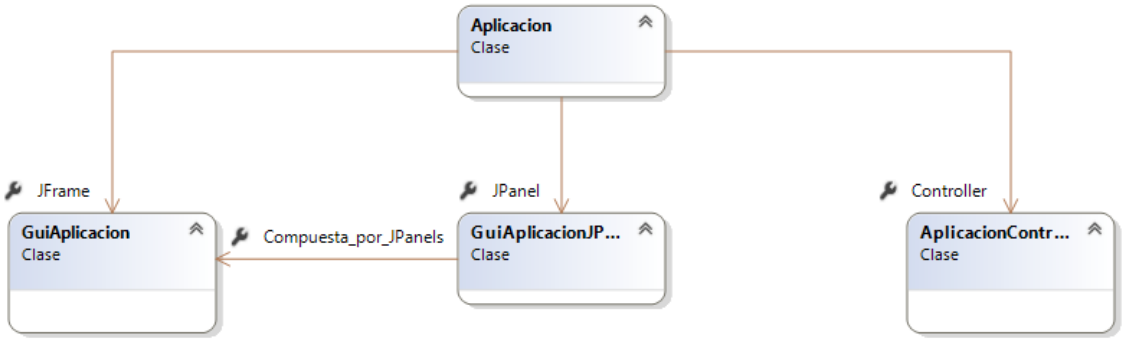

Figura 16: Diagrama de generación de código

A continuación se muestra el código fuente de la clase *Persona*.

```
@Form(name = "Persona")
public class Persona {

	private String nombre;
	private String apellido;
	@Required
	private String dni;
	@Required
	@Values(min = 2, max = 10)
	private int edad = 67;
	@Values(min = 100000000, max = 999999999)
	private int telefono;
	@ComboBox(listado = {"Madrid","Barcelona","Valencia"})
	public String[] municipio;
```

```
        private boolean enPracticas;
        private Empresa empresa;
        private Direccion direccion;

        //Constructor y main
}
```

La interfaz gráfica es la siguiente.

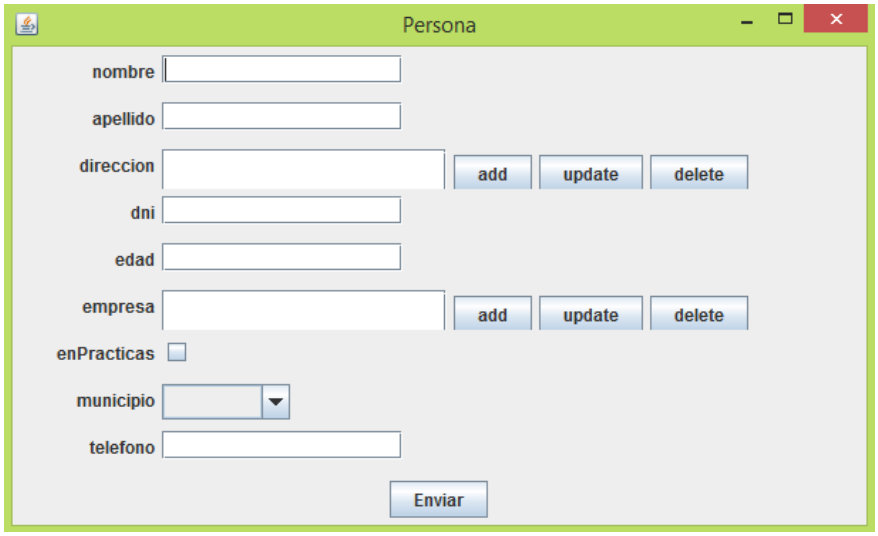

Figura 17: Ejemplo Real, interfaz gráfica de usuario de la clase Persona

A continuación se muestra el código fuente de la clase *Empresa*.

```
@Form(name = "Empresa")
public class Empresa {
        private String nombre;
        private int telefono;
        private ArrayList<Persona> lPersonas;
        private Direccion direccion;
        //Constructor y main
}
```

La interfaz gráfica generada es la siguiente.

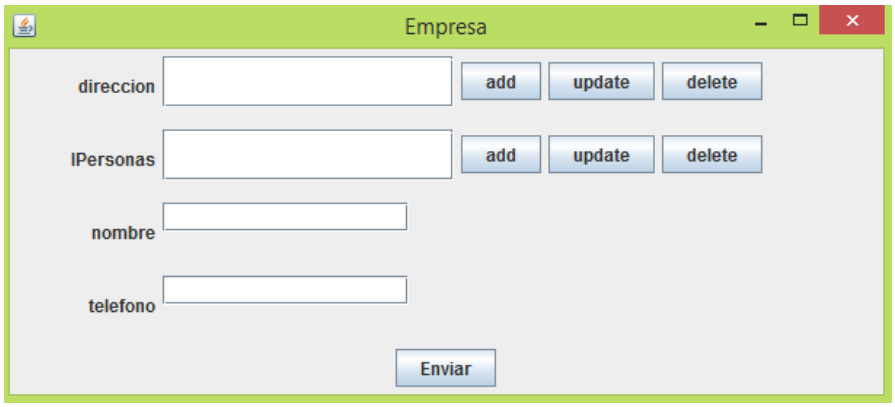

Figura 18: Ejemplo Real, interfaz gráfica de usuario de la clase Empresa

A continuación se muestra el código fuente de la clase *Direccion*.

```
@Form(name="Dirección")
public class Direccion {
```

```
        private String calle;
        private int numero;
        private int codigoPostal;
        private String poblacion;
        private String ciudad;
        //Constructor y main
}
```

La interfaz gráfica generada es la siguiente.

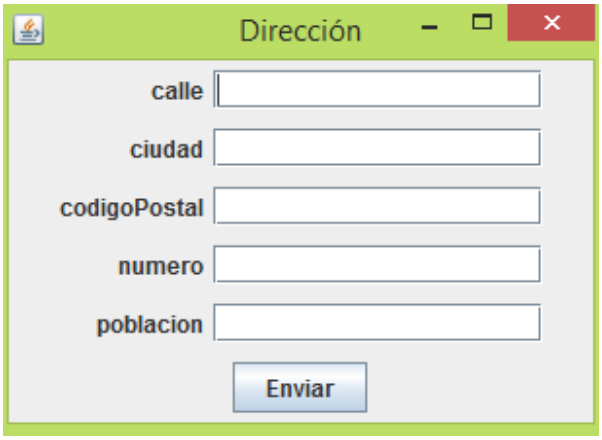

Figura 19: Ejemplo Real, interfaz gráfica de usuario de la clase Dirección

Estas son las interfaces gráficas obtenidas tras la ejecución del procesador de anotaciones.

Para poder obtener esas interfaces de usuario se han utilizados las siguientes anotaciones.

- La anotación @Form para poder generar los formularios, y con el parámetro name para que aparezca casa formulario con el nombre de la clase a la que pertenece. @Form(name = "nombre de la clase")
- La anotación @Values para dar la propiedad de un rango de valores a los campos de edad. Con los parámetros siguiente @Values (min = 18, max = 99)
- La anotación @Required para dar la propiedad de obligatorio a los campos de nombre, apellidos y calle.
- La anotación @ComboBox para poder crear un menú desplegable en el campo municipio. Con el siguiente parámetro @ComboBox (listado = {"Madrid", "Barcelona", "Valencia")

5.1.Utilización

Este procesador de anotaciones puede ser utilizado para cualquier programa desarrollado en el lenguaje Java y en el entorno de desarrollo de Eclipse. Tan solo se necesita importar al proyecto el archivo .jar del ***APT Processor*** y configurar el compilador para que utilice este procesador de anotaciones en vez del de por defecto. Con este paso se generarán estos archivos en nuestro proyecto al guardar.

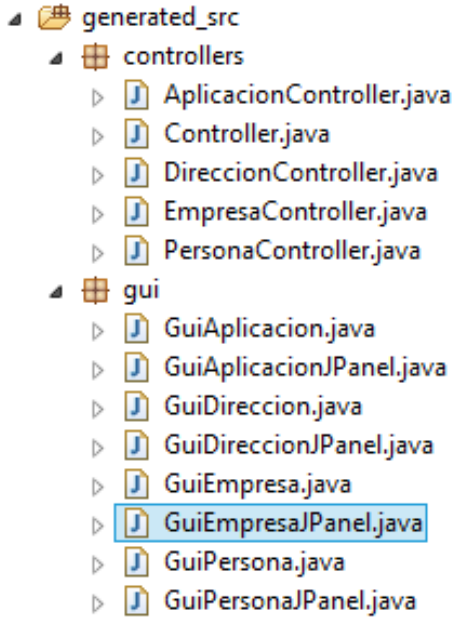

Figura 20: Ejemplo Código generado 1

A partir de estos archivos ya tenemos nuestra interfaz gráfica que se podrá editar y personalizar a gusto del usuario.

Para poder evaluar el tiempo que se puede ahorrar se mostrarán la media de líneas generadas por cada uno de los componentes.

- Para la generación de JFrame se generan de media unas 200 líneas de código.
- Para la generación de JPanel se generan de media unas 400 líneas de código
- Para la generación de los Controladores se generan de media unas 50 líneas de código.

Por tanto, el desarrollador se ahorrará una media de 650 líneas de código. Todo esto sin contar los tiempos de planificación y desarrollo de las interfaces gráficas.

6. Conclusiones

APT Processor permite la creación de interfaces gráficas por medio de las anotaciones en Java, mejorando la experiencia del usuario y aumentando su productividad.
Como se ha podido ver anteriormente ese procesador de anotaciones permite la completa integración con los proyectos en Java, tanto nuevos como existentes.

6.1. Utilización en el futuro

La utilización de las anotaciones en el lenguaje Java se está haciendo cada vez más común, dado que es un elemento que tiene mucho potencial y permite a los desarrolladores de software realizar muchas operativas que con lenguaje nativo no serían posible.
Centrándonos en este proyecto, se mostrara a continuación como se podría mejorar en el futuro su funcionalidad. Una de las utilizaciones para el futuro podría ser mejorar el ***APT Processor***, ya que con la estructura que tiene, solo es necesario cambiar la generación de código. Es decir, se podría generar código para otras librerías de interfaces gráficas en Java, o generar código en otro lenguaje, como puede ser C, C++, C#, etc. Así como la posibilidad de generar código web, en HTML y PHP.
Todo esto es posible gracias a la arquitectura modelo vista controlador que permite desligar las vistas y generarlas en cualquiera de los lenguajes nombrados anteriormente.

Anexos

Anexo 1 – Instalación APT Processor en Eclipse

Para poder utilizar esta funcionalidad se necesita tener instalado la máquina virtual de Java en una versión superior a la 1.5, y el entorno de desarrollo de Eclipse. Una vez que se cumplen los requisitos de software se importará al proyecto donde se quiere utilizar la funcionalidad del motor de anotaciones, un archivo *.jar*.
El primer paso será ir a las propiedades del proyecto y seleccionar *Java Build Path* para añadir el ***APT Processor*** a Eclipse.

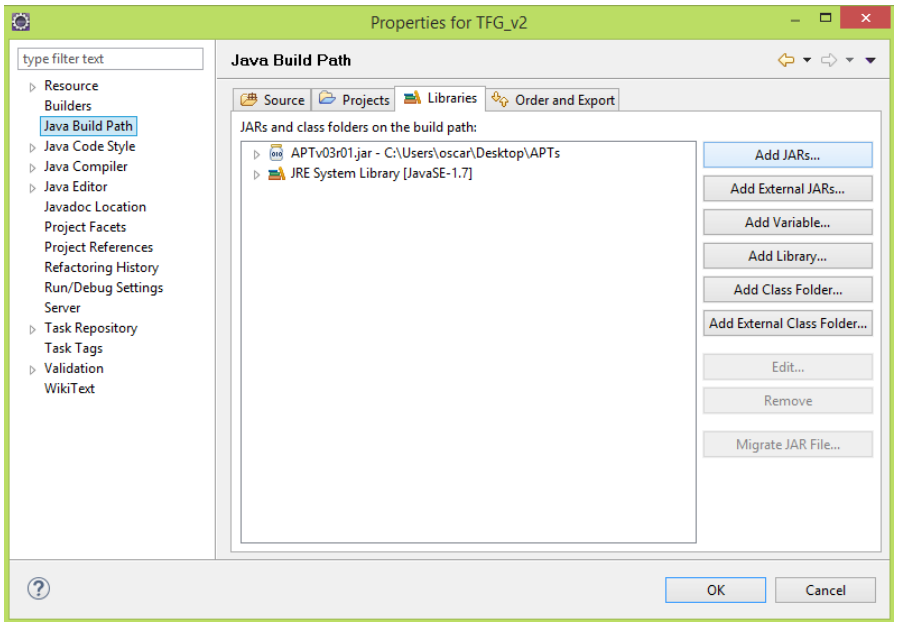

Figura 21: Propiedades del proyecto/Java Build Path

A continuación se pulsará el botón *Add JARs…* y se seleccionará el fichero del ***APT Processor***.

Una vez finalizado este paso se procederá a configurar el procesador de anotaciones de eclipse. Para ello habrá que volver a abrir las propiedades del proyecto pero esta vez seleccionar en *Java Compiler -> Annotation Precessing -> Factory Path* tal y como se muestra en la siguiente ilustración.

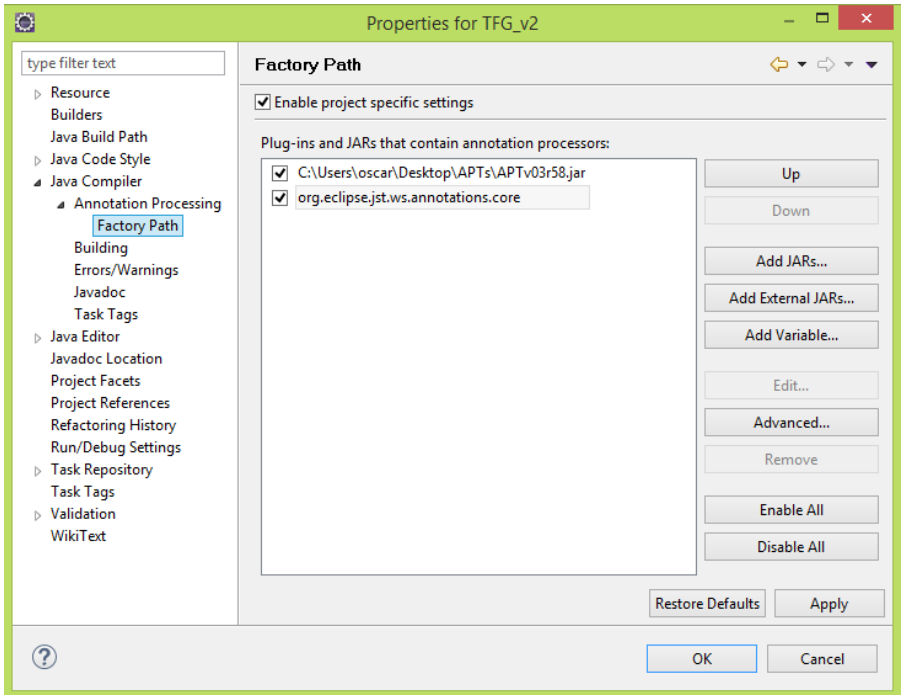

Figura 22: Propiedades del proyecto/Java Compiler/Annotation Processing

En este caso se pulsará el botón de *Add JARs…* y se seleccionará el ***APT Processor.jar***

Con estos dos sencillos pasos se podrá utilizar toda la funcionalidad descrita en este documento.

Anexo 2 – Proyecto en GitHub

Git es un software de control de versiones. Se ha utilizado para almacenar todo el proyecto del **APT Processor**. El código está disponible en:

https://github.com/OscarMelladoGonzalez/APT

Acrónimos

API *- Application Programming Interface*

IDE *- Integrated Development Environment*

JDK *- Java Development Kit*

JSR - Java Specification Request

Glosario

CheckBox - *Es un elemento de interacción de la interface gráfica de usuario.*

Framework – *Es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.*

Hibernate – *Herramienta para crear mapeos con bases de datos*

HMTL - *Lenguaje de elaboración de páginas web*

JavaScript - *Lenguaje de programación interpretado utilizado generalmente en las páginas web*

JUnit – *Conjunto de bibliotecas que son utilizadas para realizar pruebas unitarias*

Metadatos - *Literalmente "sobre datos" Son datos que describen a otros datos.*

Override - *Permite sobrescribir un método.*

Spring- *Es un framework para el desarrollo de aplicaciones*

TextBox - *Es un elemento de interacción de la interface gráfica de usuario.*

XDoclet - *Motor de código abierto, cuya función es la generación de código.*

Bibliografía

Java SE 1.5, http://www.oracle.com/technetwork/articles/javase/overview-137139.html *[Visitada por última vez 30/05/2015]*

Semantic Annotation for Java, *http://www.jot.fm/issues/issue_2010_05/column2/ [Visitada por última vez 30/05/2015]*

Metawidget, *http://metawidget.org/[Visitada por última vez 30/05/2015]*

Object 2 Gui, *https://code.google.com/p/obj2gui/ [Visitada por última vez 30/05/2015]*

Model-Driven GUI Generation, *http://java.dzone.com/articles/automatic-user-interface [Visitada por última vez 30/05/2015]*

Modelo-Vista-Controlador, *http://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador [Visitada por última vez 30/05/2015]*